Transport Layer

- Transport-layer services
- **Multiplexing and demultiplexing**
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

COMPSCI 453 Computer Networks

Professor Jim Kurose

College of Information and Computer Sciences

University of Massachusetts

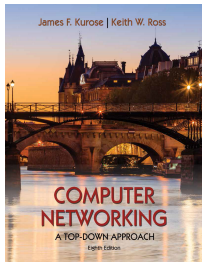
Class textbook:

Computer Networking: A Top-Down Approach (8th ed.)

J.F. Kurose, K.W. Ross

Pearson, 2020

http://gaia.cs.umass.edu/kurose_ross



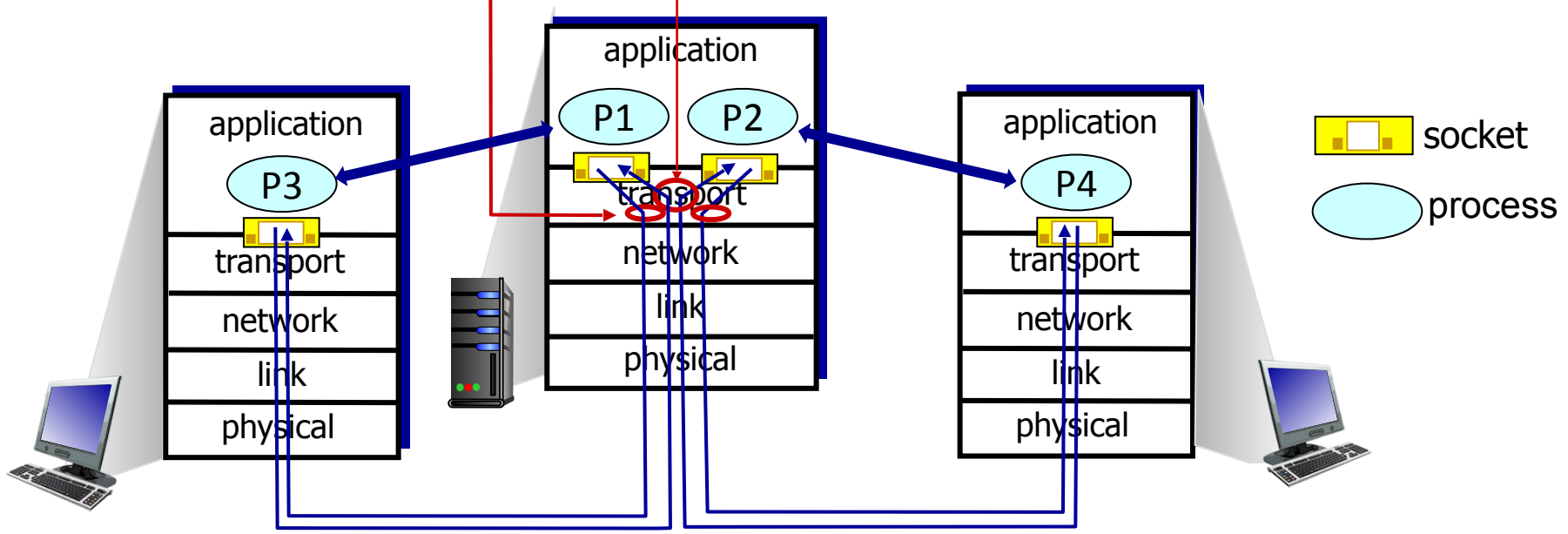
Multiplexing/demultiplexing

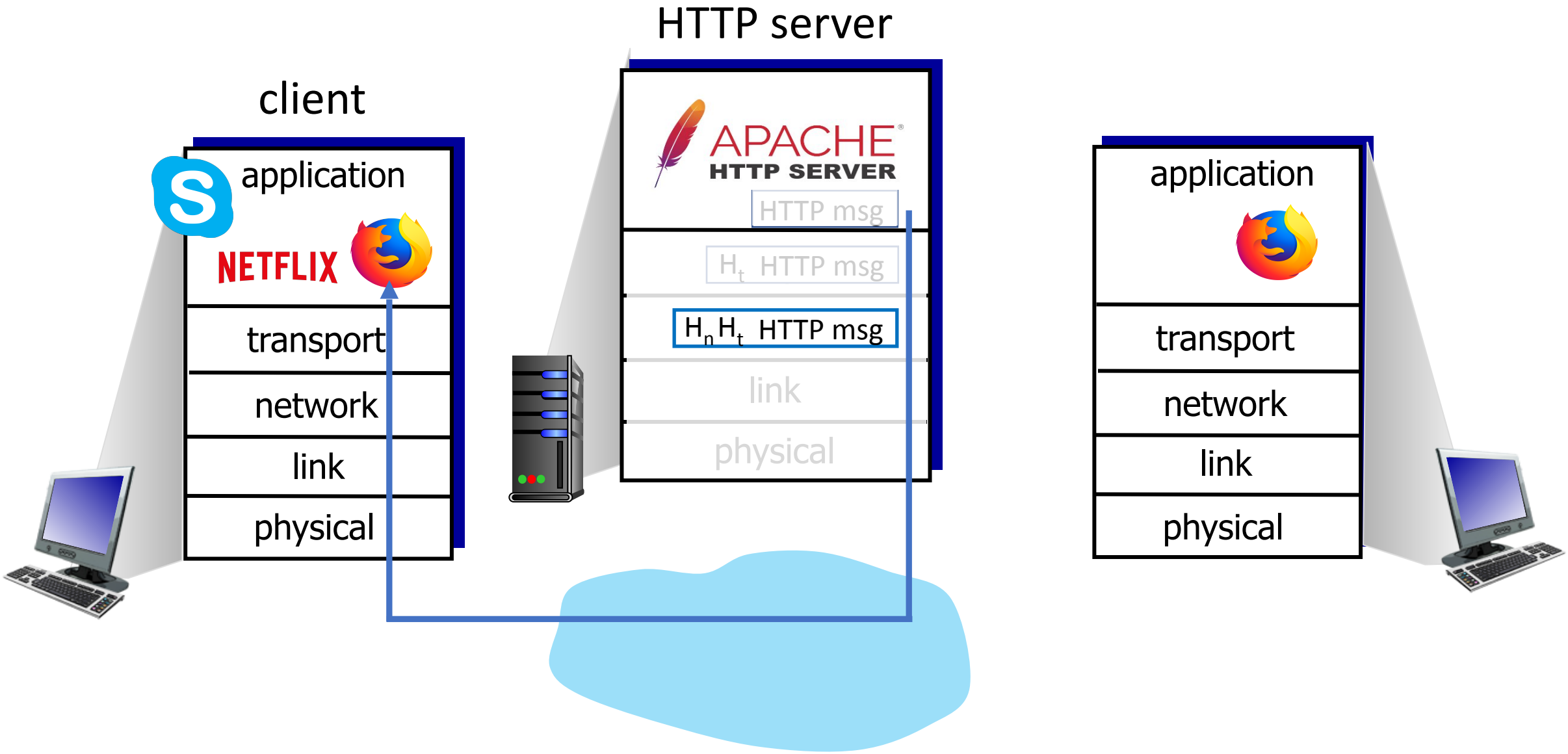
multiplexing at sender:

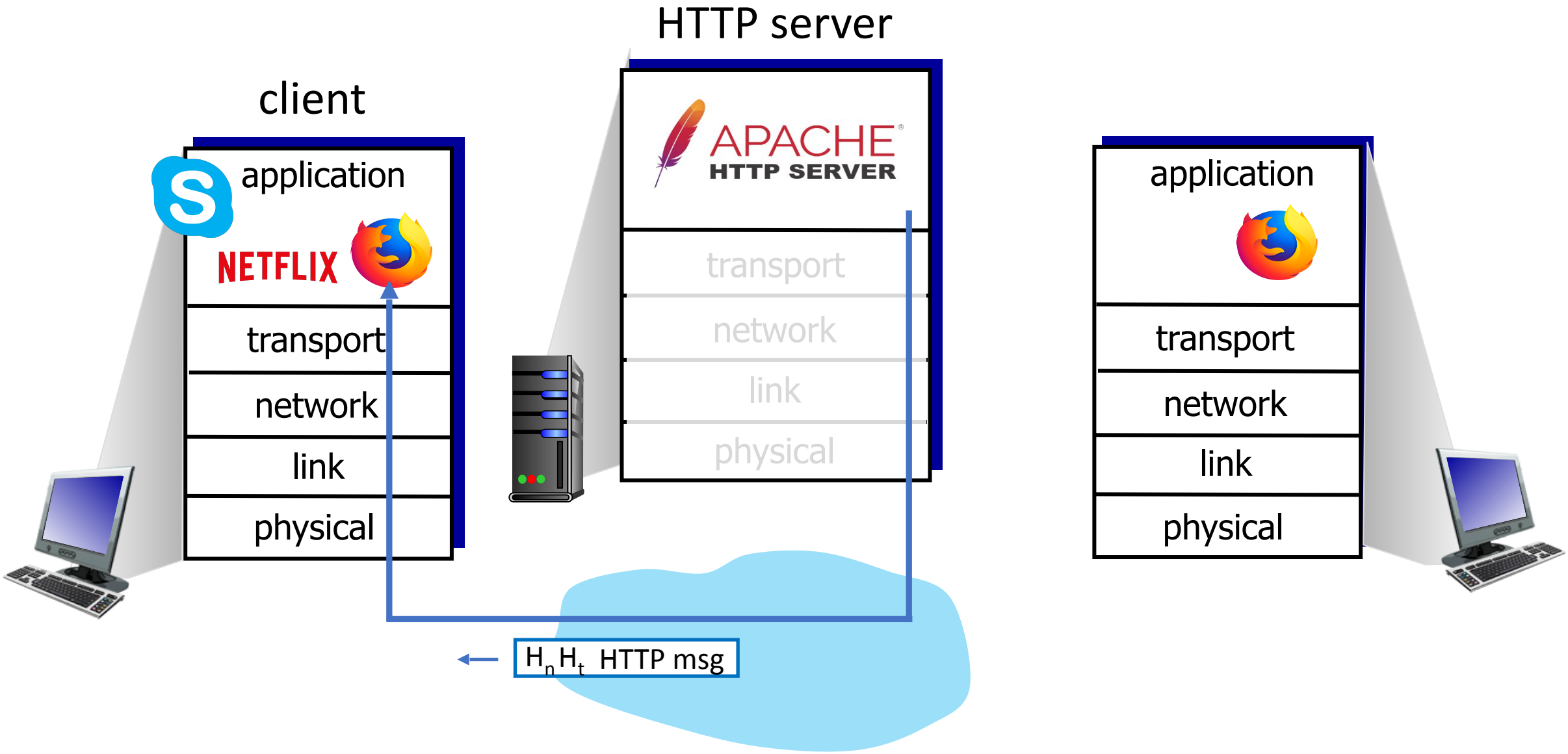
handle data from multiple sockets, add transport header (later used for demultiplexing)

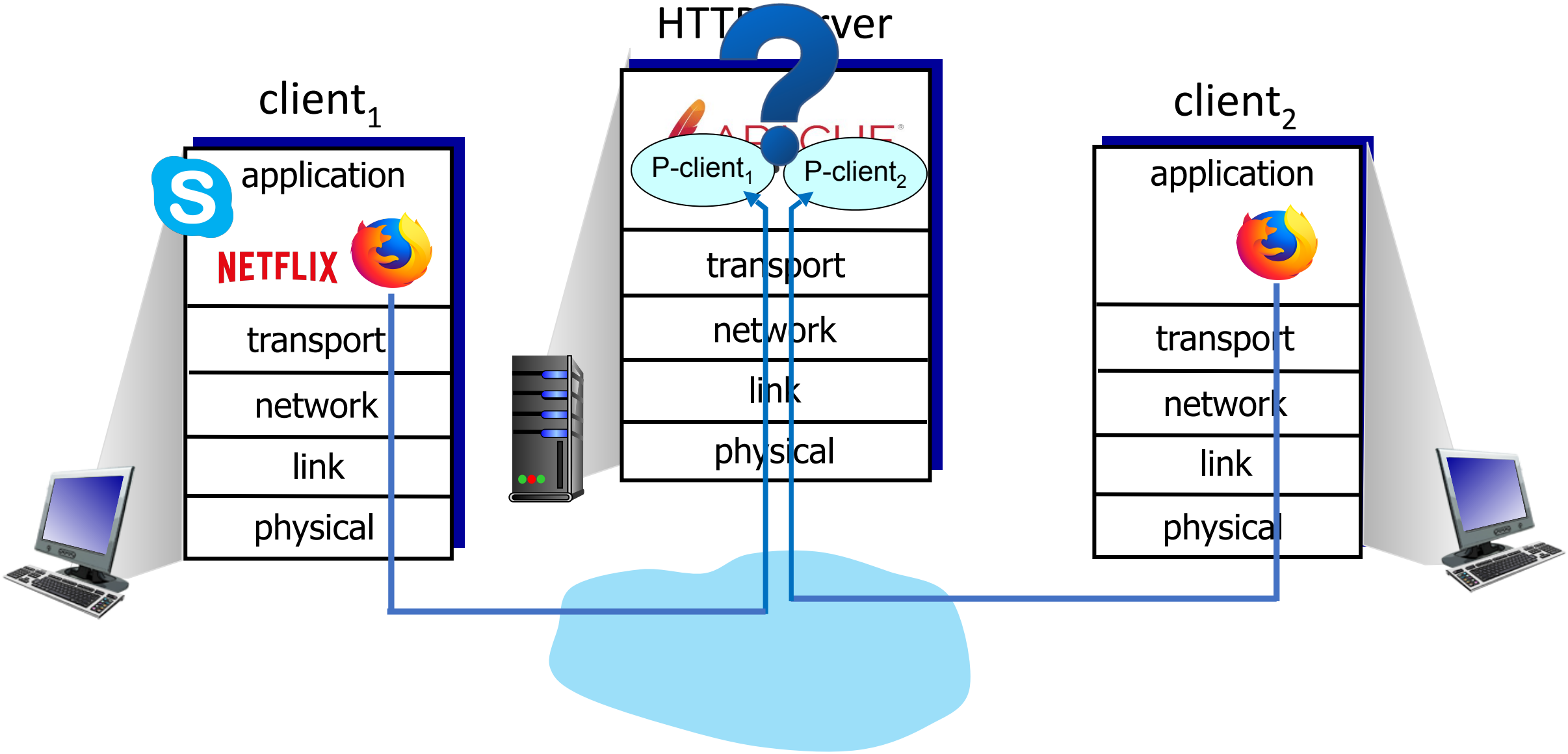
demultiplexing at receiver:

use header info to deliver received segments to correct socket



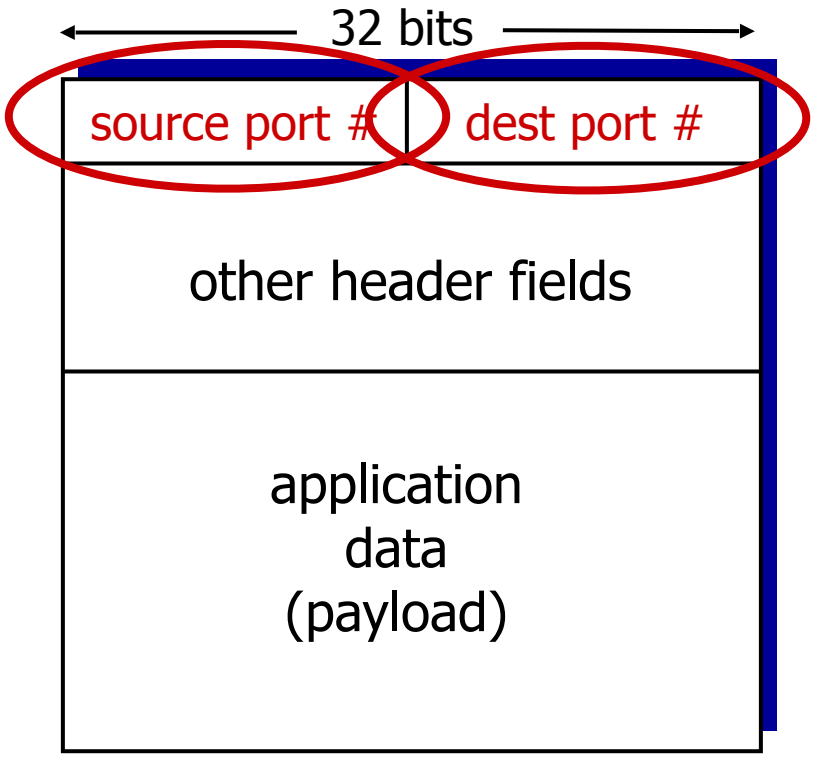

HTTP server
client
application
NETFLIX
transport
network
link
physical
APACHE
HTTP SERVER
HTTP msg
H_t HTTP msg
$H_n H_t$ HTTP msg
link
physical
application
transport
network
link
physical


HTTP server
client
application
NETFLIX
transport
network
link
physical
APACHE
HTTP SERVER
transport
network
link
physical
application
transport
network
link
physical
$H_n H_t$ HTTP msg


client1
application
NETFLIX
transport
network
link
physical
HTTP server
P-client1
P-client2
transport
network
link
physical
client2
application
transport
network
link
physical

How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number
- host uses *IP addresses & port numbers* to direct segment to appropriate socket



TCP/UDP segment format

Connectionless demultiplexing

Recall:

- when creating socket, must specify *host-local* port #:

```
DatagramSocket mySocket1 =
  new DatagramSocket(12534);
```

- when creating datagram to send into UDP socket, must specify
  - destination IP address
  - destination port #

when receiving host receives *UDP* segment:

- checks destination port # in segment
- directs UDP segment to socket with that port #

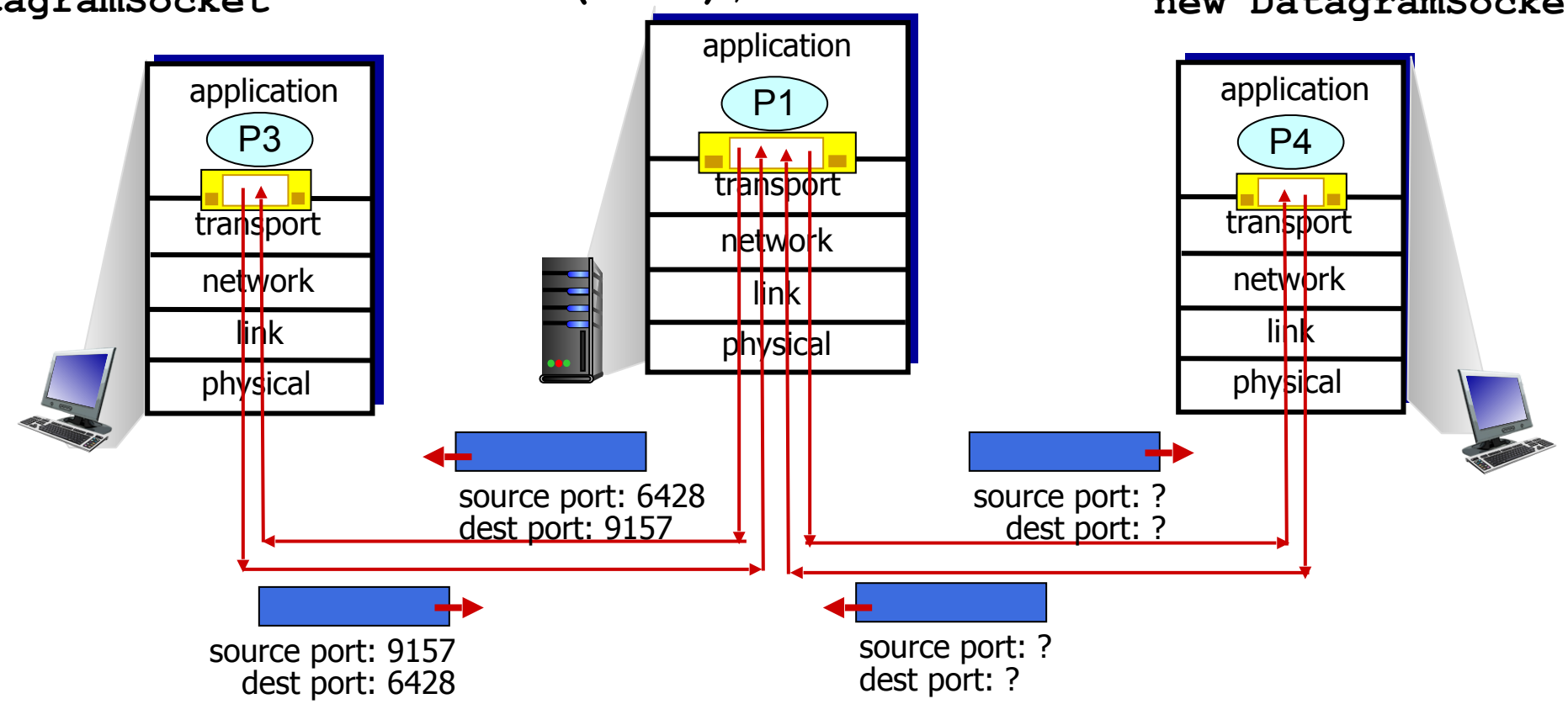
IP/UDP datagrams with *same dest. port #,* but different source IP addresses and/or source port numbers will be directed to *same socket* at receiving host

Connectionless demultiplexing: an example

```
DatagramSocket mySocket2 =
 new DatagramSocket
 (9157);
```

```
DatagramSocket
 serverSocket = new
 DatagramSocket
 (6428);
```

```
DatagramSocket mySocket1 =
 new DatagramSocket (5775);
```

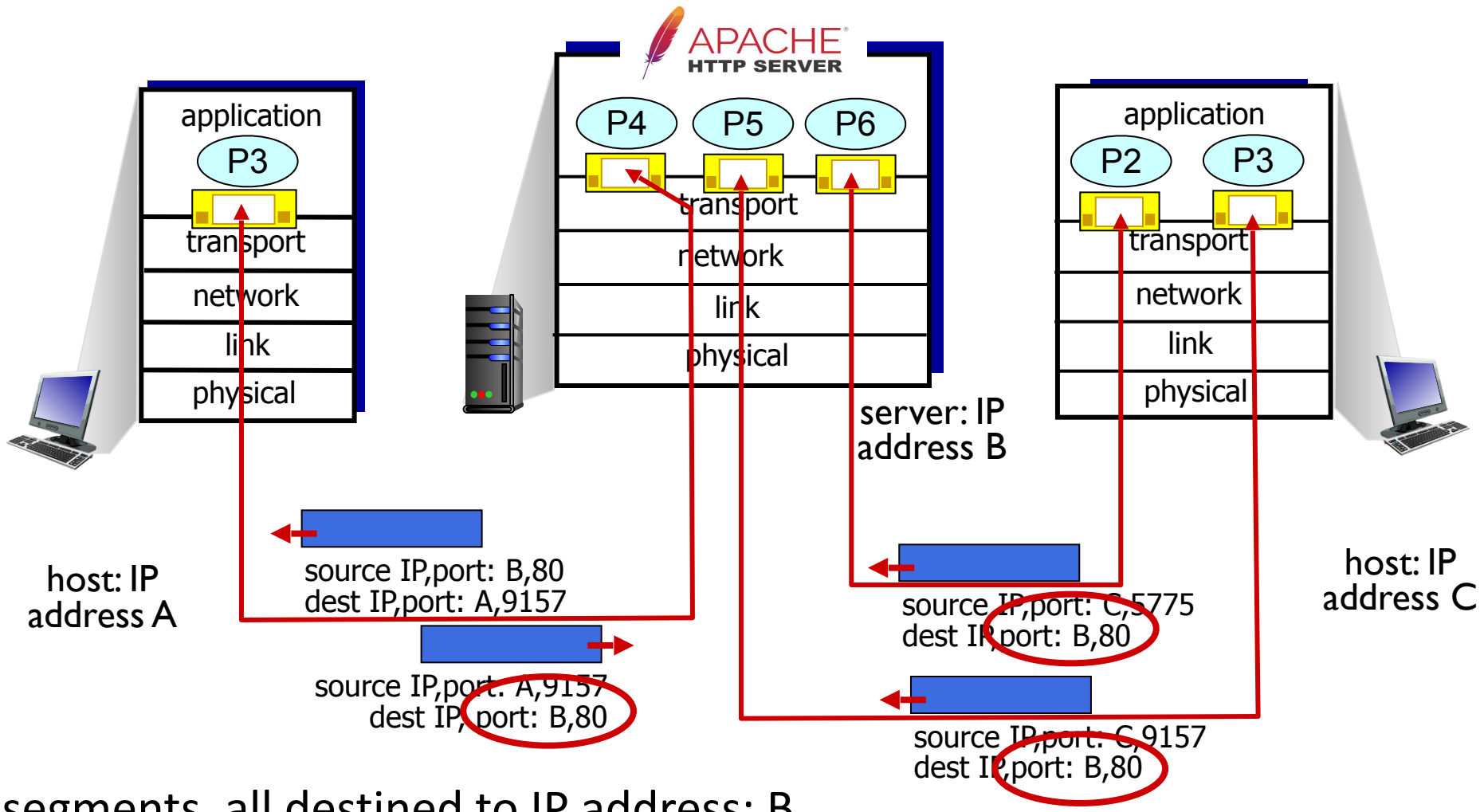


Connection-oriented demultiplexing

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses *all four values (4-tuple)* to direct segment to appropriate socket

- server may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
  - each socket associated with a different connecting client

Connection-oriented demultiplexing: example



Three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

Summary

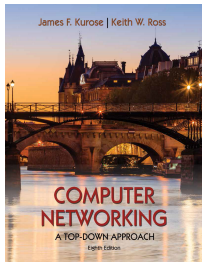
- Multiplexing, demultiplexing: based on segment, datagram header field values
- **UDP:** demultiplexing using destination port number (only)
- **TCP:** demultiplexing using 4-tuple: source and destination IP addresses, and port numbers
- Multiplexing/demultiplexing happen at *all* layers

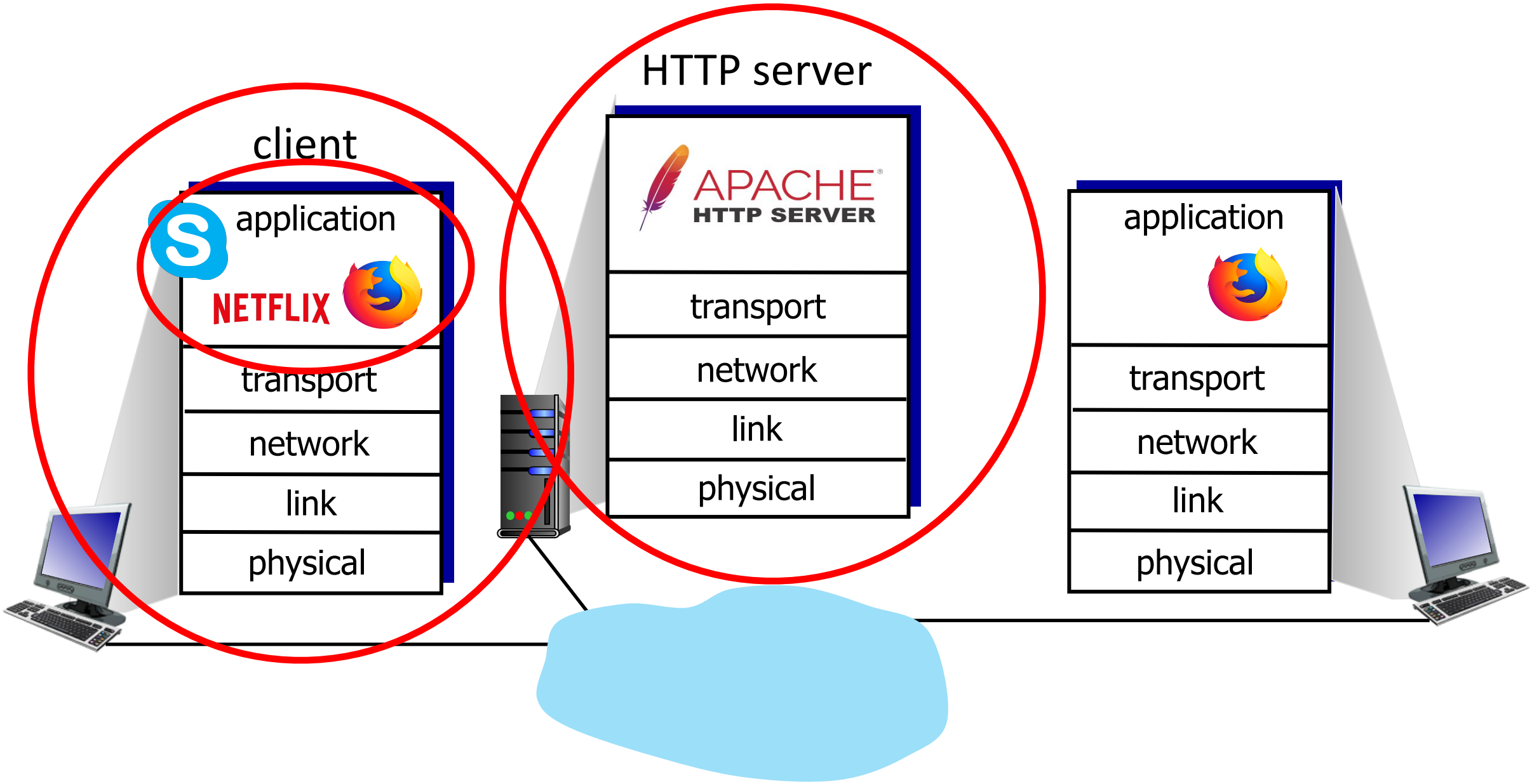
Transport Layer

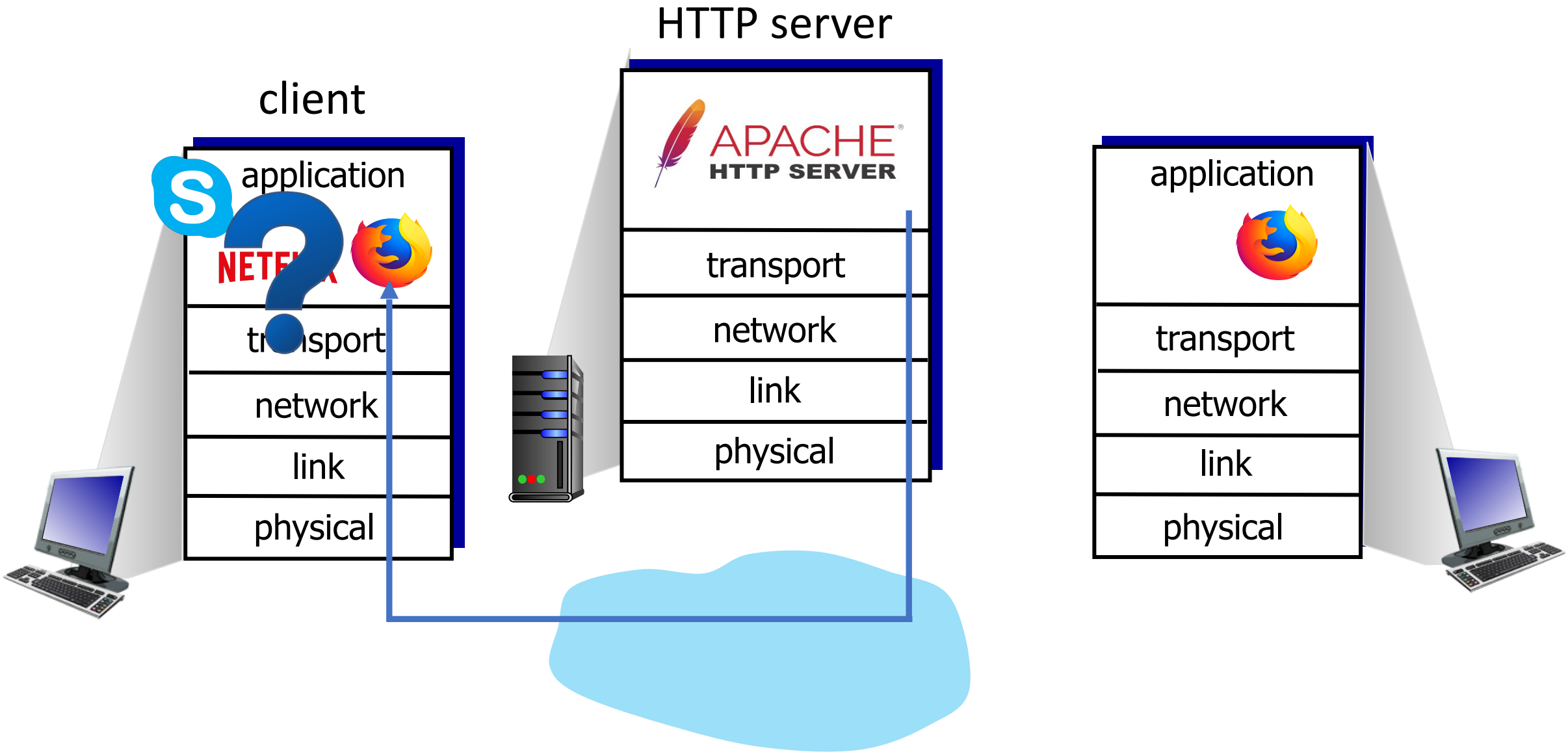
- Transport-layer services
- **Multiplexing and demultiplexing**
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

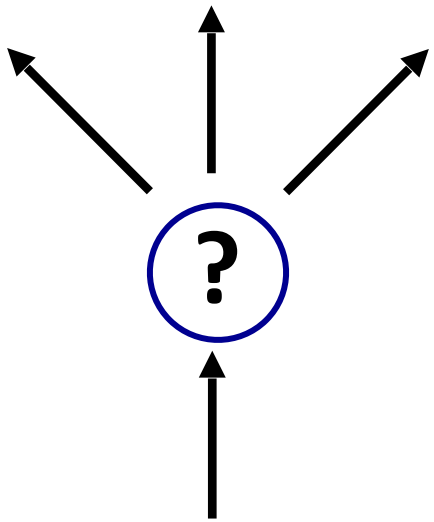
COMPSCI 453 Computer Networks
Professor Jim Kurose
College of Information and Computer Sciences
University of Massachusetts

Class textbook:
Computer Networking: A Top-Down Approach (8th ed.)
J.F. Kurose, K.W. Ross
Pearson, 2020
http://gaia.cs.umass.edu/kurose_ross

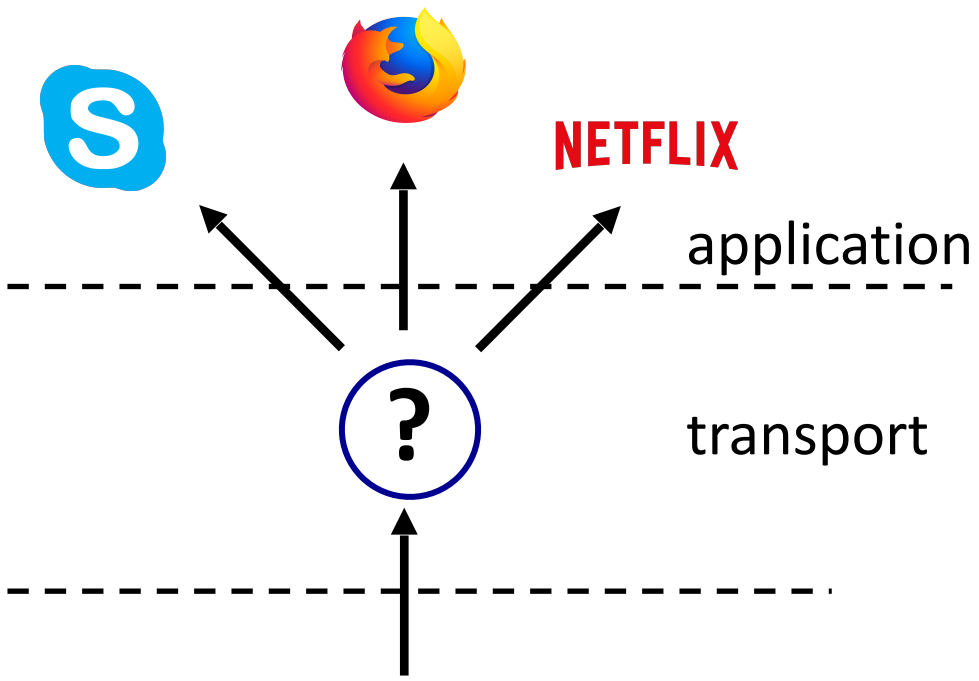



client
application
NETFLIX
transport
network
link
physical
HTTP server
APACHE
HTTP SERVER
transport
network
link
physical
application
transport
network
link
physical

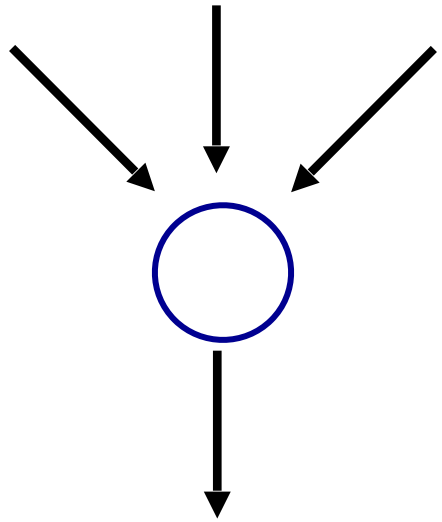

HTTP server
client
application
transport
network
link
physical
APACHE
HTTP SERVER
transport
network
link
physical
application
transport
network
link
physical



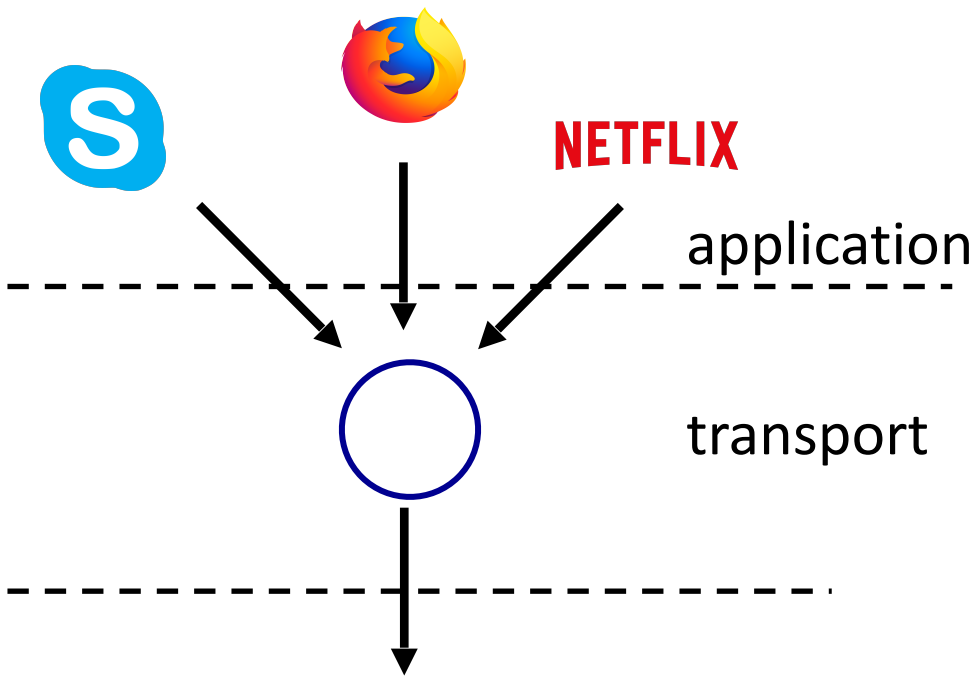
de-multiplexing



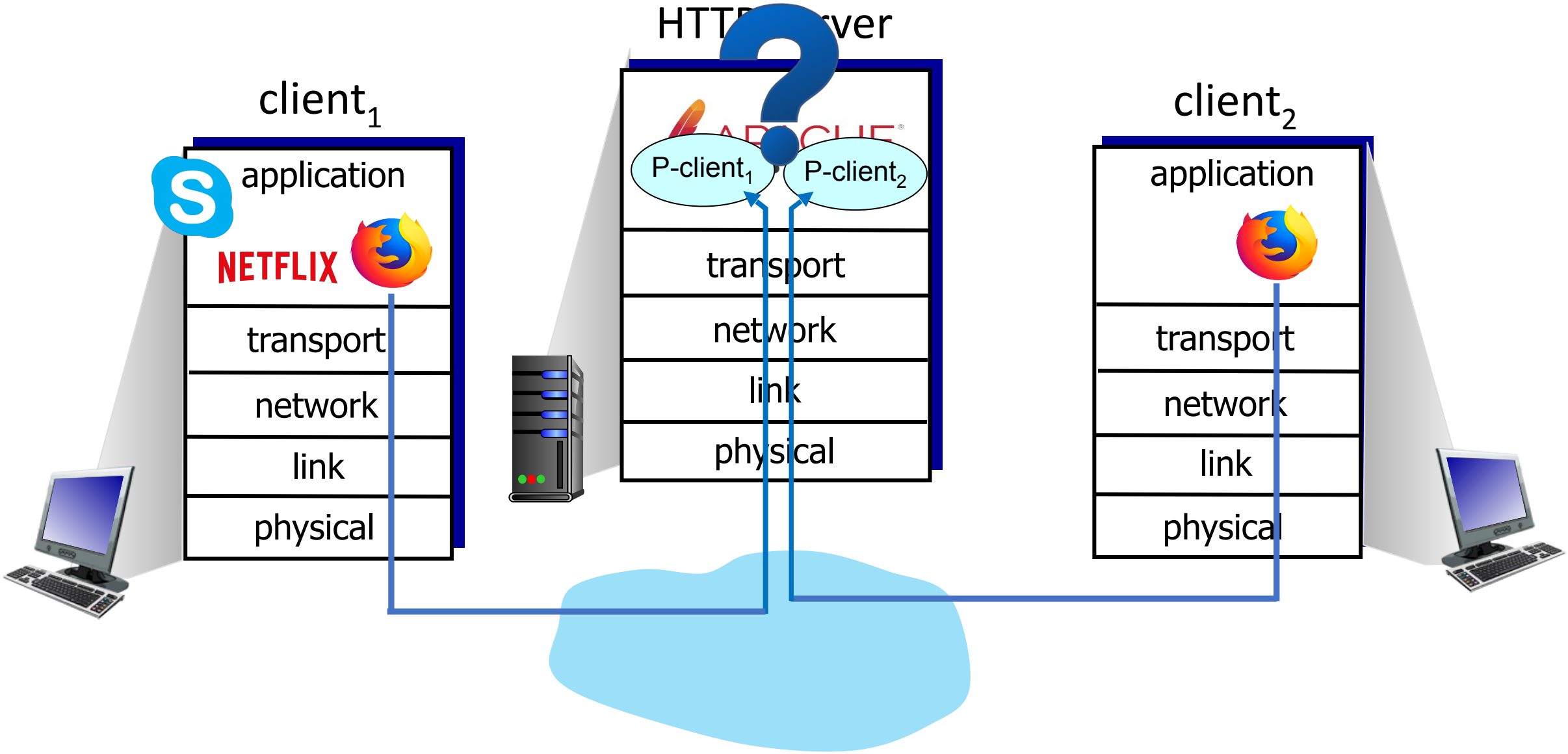
de-multiplexing



multiplexing



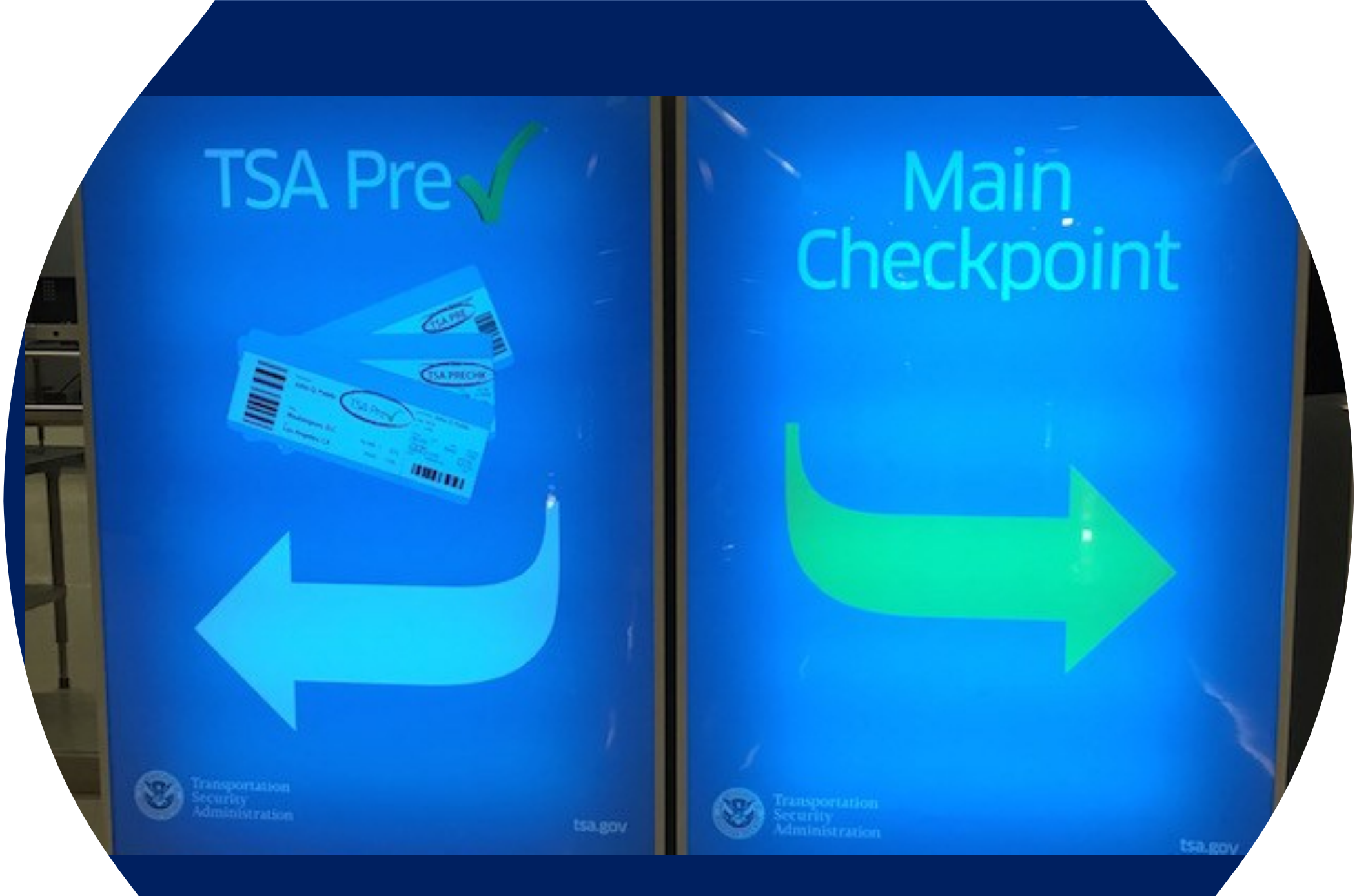
multiplexing


client1
application
NETFLIX
transport
network
link
physical
HTTP server
P-client1
P-client2
transport
network
link
physical
client2
application
transport
network
link
physical

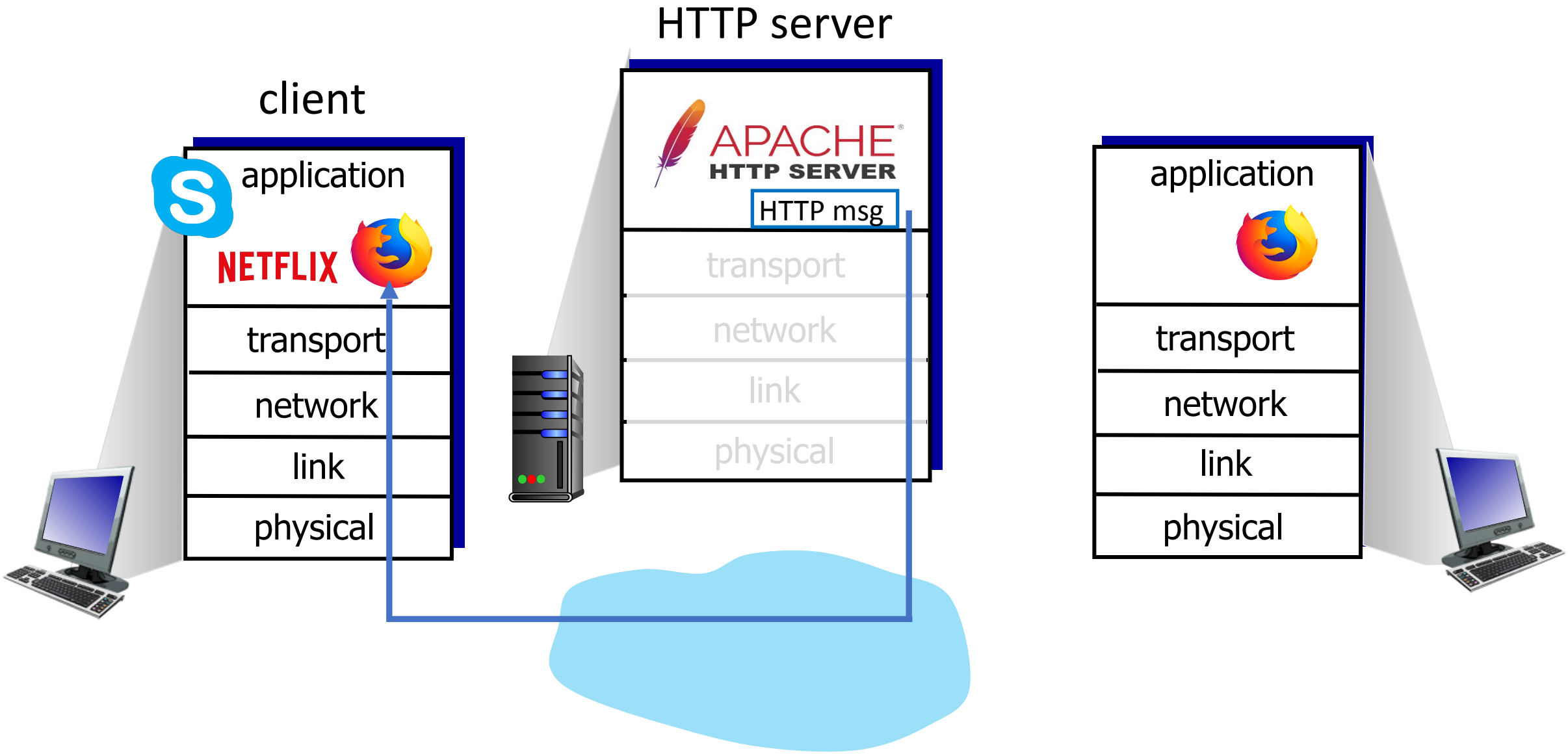

Multiplexing

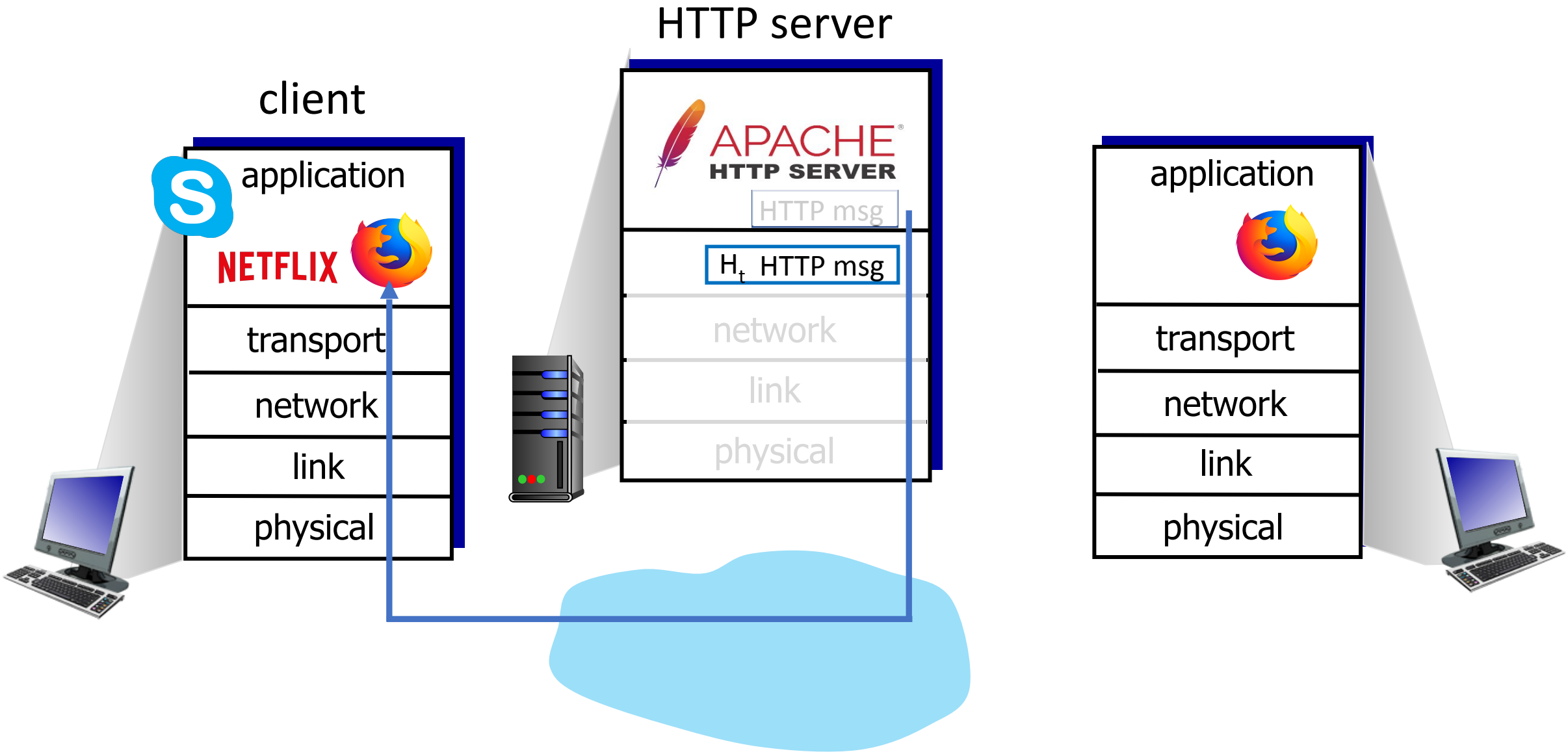
Demultiplexing


AIRFRANCE
ECONOMY
SKYTEAM
AIRFRANCE
SKY
PRIORITY™
SKYTEAM


TSA Pre✓
TSA PRE
TSA PRECHK
TSA Pre✓
Transportation Security Administration
tsa.gov
Main Checkpoint
Transportation Security Administration
tsa.gov


SKY PRIORITY
DELTA
LG
1
ZONE
DELTA
2
ZONE
3
ZONE
B14


HTTP server
client
application
NETFLIX
transport
network
link
physical
APACHE
HTTP SERVER
HTTP msg
transport
network
link
physical
application
transport
network
link
physical


HTTP server
client
application
NETFLIX
transport
network
link
physical
APACHE
HTTP SERVER
HTTP msg
H_t HTTP msg
network
link
physical
application
transport
network
link
physical